Lógica de programação

Profº Jânio Eduardo

janio.vasconcellos@gmail.com

Roteiro

- Objetivo geral
- Introdução Lógica de programação
  - Construção de algoritmos
  - Algoritmos com estruturas de decisão
  - Algoritmos com estruturas de repetição
  - Exercícios
- Resumo da aula

Objetivo geral

Aprender a montar algoritmos de decisão e repetição, variáveis e constantes

Professor Jânio Eduardo

Construção de algoritmos

- A lógica de programação é essencial para pessoas que desejam trabalhar com desenvolvimento de programas para computadores. Lógica de programação pode ser definida como um conjunto de técnicas para encadear pensamentos a fim de atingir determinado objetivo.

O objetivo fundamental de toda programação é construir **algoritmos**. Mas, afinal, o que é um algoritmo?

- Em outras palavras, quando criamos um algoritmo, apenas apontamos uma sequência de atividades que levam à solução de um problema. Até mesmo as soluções para os problemas cotidianos mais simples podem ser descritas por sequências lógicas de atividades, ou seja, por algoritmos:

Problema: Trocar uma lâmpada

- **Sequência de Passos para Solução:**
  - **1.** Pegue uma escada;
  - **2.** Posicione a escada embaixo da lâmpada;
  - **3.** Pegue uma lâmpada nova;
  - **4.** Suba na escada;
  - **5.** Retire a lâmpada velha;
  - **6.** Coloque a lâmpada nova.

Agora é com vocês

- **Atividade** – Escreva um algoritmo (sequência de passos) para trocar um
- pneu de um carro.
- **Atividade** – Descreva um algoritmo que defina como fazer um bolo.
- **Atividade** – Descreva um algoritmo que defina como preparar um ovo
- frito.

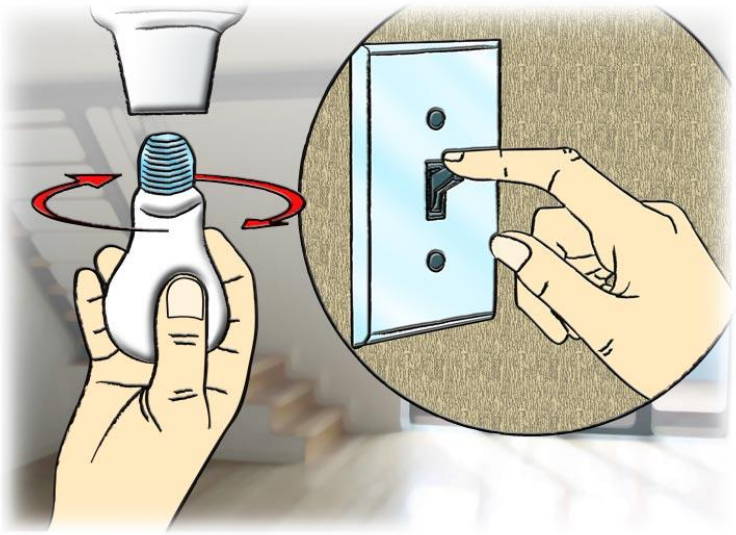
Algoritmos com estruturas de decisão

- Os algoritmos que construímos até agora apresentam uma sequência de passos que devem ser seguidos para atingir um objetivo bem definido. Note que todos os passos dos algoritmos devem ser executados a fim de que o objetivo seja alcançado.

- Observação
  - Porém, há algoritmos nos quais a execução de alguns passos pode depender de **decisões** a serem tomadas. Dessa forma, algum fato indicará se um ou mais passos do algoritmo serão executados ou não.

Algoritmos com estruturas de decisão

- Por exemplo, o nosso primeiro algoritmo define uma sequência de passos para trocar uma lâmpada. Em momento algum perguntamos se a lâmpada está queimada. Simplesmente trocamos a lâmpada sem fazer qualquer teste.



Teste

- Para resolver esse problema, podemos acrescentar ao nosso algoritmo um **teste** que verifique se a lâmpada deve ser trocada:
- **1.** Ligue o interruptor
- **2. Se** a lâmpada não acender:
- **2.1.** Pegue uma escada;
- **2.2.** Posicione a escada embaixo da lâmpada;
- **2.3.** Pegue uma lâmpada nova;
- **2.4.** Suba na escada;
- **2.5.** Retire a lâmpada velha;
- **2.6.** Coloque a lâmpada nova.

Algoritmos com estruturas de repetição

- Note que, apesar de nosso novo algoritmo estar verificando a necessidade de trocar a lâmpada antes de fazê-lo, em momento algum verificamos se a lâmpada nova que foi instalada funciona. Assim, vamos tentar alterar o nosso algoritmo a fim de garantir que ao fim de sua execução tenhamos uma lâmpada funcionando. Para isso, vamos incluir um novo teste em seu final:

Algoritmos com estruturas de repetição

- **1.** Ligue o interruptor
- **2.** Se a lâmpada não acender:
  - **2.1.** Pegue uma escada;
  - **2.2.** Posicione a escada embaixo da lâmpada;
  - **2.3.** Pegue uma lâmpada nova;
  - **2.4.** Suba na escada;
  - **2.5.** Retire a lâmpada velha;
  - **2.6.** Coloque a lâmpada nova;
  - **2.7.** Se a lâmpada não acender:
    - **2.7.1.** Retire a lâmpada;
    - **2.7.2.** Coloque uma outra lâmpada;
    - **2.7.3.** Se a lâmpada ainda não acender:
      - **2.7.3.1.** Retire a lâmpada;
      - **2.7.3.2.** Coloque uma outra lâmpada;
      - **(Até quando ficaremos nesses testes???)**

Algoritmos com estruturas de repetição

- Pelo nosso novo algoritmo, caso a nova lâmpada não acenda, devemos trocá-la novamente e repetir esse procedimento indefinidamente até que uma lâmpada funcione. Note que não sabemos quantas vezes teremos de repetir o teste até acharmos uma lâmpada que funcione.
- Em casos como esse, devemos utilizar estruturas de repetição. Essas estruturas definem um fluxo de ações que se repete enquanto uma determinada situação acontece.

Dessa forma, substituímos nossa sequência indefinida de estruturas de decisão por uma estrutura de repetição:

- **1.** Ligue o interruptor;
- **2.** Se a lâmpada não acender:
  - **2.1.** Pegue uma escada;
  - **2.2.** Posicione a escada embaixo da lâmpada;
  - **2.3.** Pegue uma lâmpada nova;
  - **2.4.** Suba na escada;
  - **2.5.** Retire a lâmpada velha;
  - **2.6.** Coloque a lâmpada nova.
  - **2.7. Enquanto a lâmpada não acender:**
    - **2.7.1.** Retire a lâmpada;
    - **2.7.2.** Coloque uma outra lâmpada.

Atividades

- Acesse nosso git para baixar a lista de execicios

Conceitos básicos para a construção de algoritmos para computadores

Variáveis

- O primeiro passo para que um programa seja executado em um computador é o carregamento desse programa para a memória. A **memória** é utilizada para armazenar tanto as instruções dos programas quanto os dados utilizados pelos mesmos. Qualquer programa, para ser executado, tem de estar na memória (FARRER, 1999).
- Ao desenvolvermos nossos algoritmos, frequentemente precisamos armazenar dados referentes ao problema, como um nome, um número ou mesmo o resultado de uma operação. Mas, para armazenar esses dados, precisamos solicitar ao computador que ele reserve uma área da memória para nosso uso. A forma de solicitar ao computador que reserve memória é chamada de **declaração de variáveis.**

Sintaxe:

Tipo_da_variável nome_variável = valor_recebido;

Regras para nomear variáveis

FBr
Faculdade Brasília

Quadro 2.1: Regras para nomear variáveis	
REGRA	**EXEMPLO**
Inicie sempre por um caractere alfabético, nunca por um número.	Nome (correto) - 1nome (errado)
Não utilize caracteres especiais como " , ( ) / * ; + .	Nome(M); N*B
Não coloque espaços em branco ou hífen entre nomes.	salario-bruto
Utilize, se necessário, *underline*.	salario_bruto
Crie suas variáveis com nomes sugestivos.	Se vai guardar salário de funcionários, dê à variável o nome **salario**.

Tipos de variáveis

- **Tipo inteiro:** Declararemos variáveis do tipo **numérico inteiro** quando precisarmos armazenar valores inteiros, positivos ou negativos (0, 1, 5, 7, -10, -5...). Por exemplo, se precisarmos de uma variável para armazenar o número de filhos de um funcionário, o tipo ideal para essa variável seria **inteiro**.
- **Tipo real**: Declararemos variáveis do tipo **numérico real** para armazenar valores reais; em outras palavras, valores com ponto decimal (5.7, 3.2, -8.5). Esse seria o tipo ideal para armazenar, por exemplo, o salário de funcionários.
- **Tipo caractere**: Declararemos variáveis do tipo **literal caractere** para armazenar um único caractere, que pode ser uma letra ou um símbolo. Por exemplo, para identificar o sexo do indivíduo, armazenaremos apenas o caractere 'F' ou 'M'.
- **Tipo cadeia**: Declararemos variáveis do **tipo literal cadeia** para armazenar uma sequência de caracteres, ou seja, uma palavra, uma mensagem, um nome. Assim, se precisarmos de uma **variável** para armazenar o nome de uma pessoa, esse seria o tipo ideal.
- **Tipo lógico:** Declararemos variáveis do tipo lógico para armazenar valores lógicos. O valor de variáveis desse tipo será sempre VERDADEIRO ou FALSO.

Tipo de variáveis em C

Tipo	Descrição	Exemplo
Int	usado para armazenar números inteiros.	int idade = 27;
float e double	usados para armazenar números decimais.	float altura = 1.75; double peso = 68.5;
Caractere (char)	usado para armazenar caracteres individuais.	char letra = 'A';
Cadeia de caracteres (string)	usado para armazenar uma sequência de caracteres	char nome[] = "Maria";
Booleano (bool):	usado para armazenar valores verdadeiro ou falso.	bool ehMaiorDeIdade = true;

Atividades

- Aprendemos algumas regras que devem ser seguidas para dar nomes a variáveis. Assinale os nomes de variáveis que obedecem a essas regras:

a) ( ) nome
b) ( ) telefone-celular
c) ( ) nome+sobrenome
d) ( ) 2taxa
e) ( ) telefone_celular
f) ( ) conta1

Atividades

- Para cada valor dado abaixo foi definido um tipo de variável. Marque os pares “valor e tipo” definidos corretamente:

  a) ( ) valor = 2.5 tipo= real

  b) ( )valor = ‘F’ tipo= inteiro

  c) ( ) valor = -2 tipo= inteiro

  d) ( ) valor = ‘M’ tipo= caractere

  e) ( ) valor = 5 tipo= cadeia

  f) ( ) valor = -10.35 tipo= real

  g) ( ) valor = 38 tipo= real

  h) ( )valor = ‘Jose’ tipo= cadeia

  i) ( ) valor = 135 tipo= inteiro

  j) ( )valor = 7.5 tipo= inteiro

Atividade

- Você está fazendo um algoritmo para calcular a média dos alunos a partir das notas de duas provas. Assim, precisará de três variáveis: uma para a nota da primeira prova, uma para a nota da segunda prova e uma para a média. Segundo as normas da instituição, as notas das provas devem ser números inteiros de 0 a 10. Já para a média podem ser atribuídos valores com casas decimais. Utilizando a sintaxe de declaração de variáveis em C e as regras para definição de tipos e de nomes, indique como você declararia essas 3 variáveis. **Dica: lembre-se de escolher nomes sugestivos para as variáveis.**

Constantes

- Como aprendemos, o valor de uma variável pode ser alterado ao longo de seu algoritmo. Mas, às vezes, precisamos armazenar valores que não se alteram. Para isso existem as constantes.

- **As constantes são criadas obedecendo às mesmas regras de nomenclatura já vistas em variáveis.** Diferem apenas no fato de armazenar um valor constante, ou seja, que não se modifica durante a execução de um programa.

- A sintaxe para a declaração de constantes em C é dada abaixo:

- *Em C, as constantes podem ser declaradas usando a diretiva **#define** ou a palavra-chave '**const**'. A diretiva #**define** é usada para definir constantes de pré-processamento, enquanto a palavra-chave **const** é usada para definir constantes de tempo de compilação.*

```
#define nome_da_variável valor
#define PI 3.14159

const double PI    3.14159
```

Se tiver tempo vamos seguir com a entrada de dados

```c
#include <stdio.h> //biblioteca que fornece entrada e saída dos dados

int main() {//função principal da linguagem C


    int  nota1, nota2; //declara as variáveis que recebe a nota
    Float media;

    printf("Digite a primeira nota: ");

    scanf("%f", &nota1); //recebe a entrada do valor para nota 01

    printf("Digite a segunda nota: ");

    scanf("%f", &nota2); //recebe a entrada do valor para nota 02

    media = (nota1 + nota2) / 2; //calcula a média da nota

     printf("A media das notas eh: %.2f", media); //exibe o valor média da nota no terminal

     return 0;
}
```

Resumo da aula

- Aprendemos:
  - Utilizar logica com condição
  - Utilizar lógica com repetição
  - Aplicar algoritmo para declarar variáveis
  - Aplicar algoritmo para declarar constantes
- Próxima aula vamos entender mais de lógica para aplicarmos no código;

Obrigado!!!

Profº Jânio Eduardo

janio.vasconcellos@gmail.com